Computer Organization

Direct Memory Access

Dongup Kwon (dongup@snu.ac.kr)
Pyeongsu Park (pyeongsu@snu.ac.kr)
High Performance Computer System (HPCS) Lab

May 28, 2017

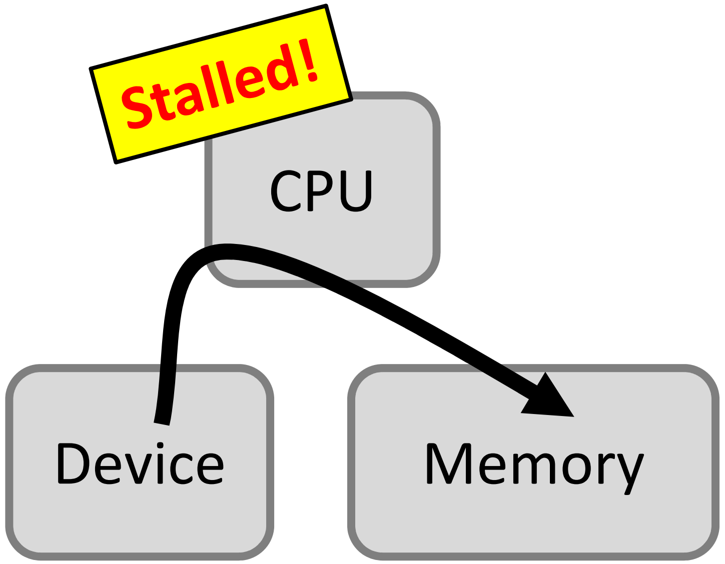
Goals

- Understand DMA and its impact on performance
- Understand how devices communicate with the CPU and the memory
- Implement a simplified DMA logic on your pipelined CPU with cache

Peripheral Devices

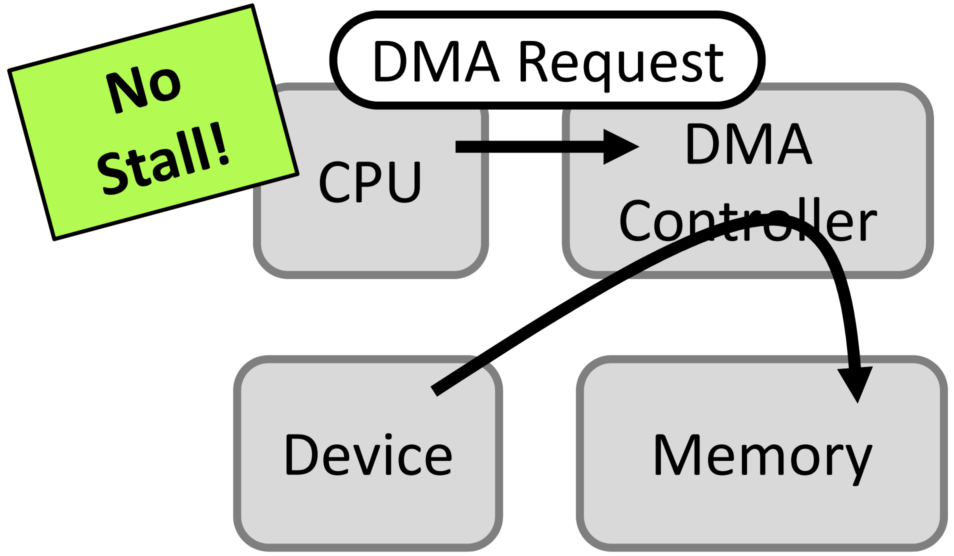
- A computer has lots of peripheral devices (e.g., HDD, GPU, network interface card (NIC)) for various purpose (e.g., transfer data to the memory).
- As a CPU cannot use the device’s functionality directly, there should be a communication mechanisms.
    - Interrupt is the heart of this mechanism.
    - When a device needs a help from a CPU, the device uses interrupt to request certain operations (e.g., memory write).
    - When the CPU gets the interrupt, it handles it.

Naïve Data Transfer



- The CPU must perform load/store instructions when transferring data from the I/O device to the memory.
  - The CPU reads data from the I/O device and write them to the memory.
  - It consumes a large number of CPU cycles.

Direct Memory Access (DMA)



- DMA decouples I/O-to-memory data transfers from other instructions.
  - The CPU requests I/O-to-memory data transfers to the DMA controller.
  - The DMA controller is responsible for transferring data from the device to the memory.
  - The CPU executes other instructions while the DMA controller transfers data.

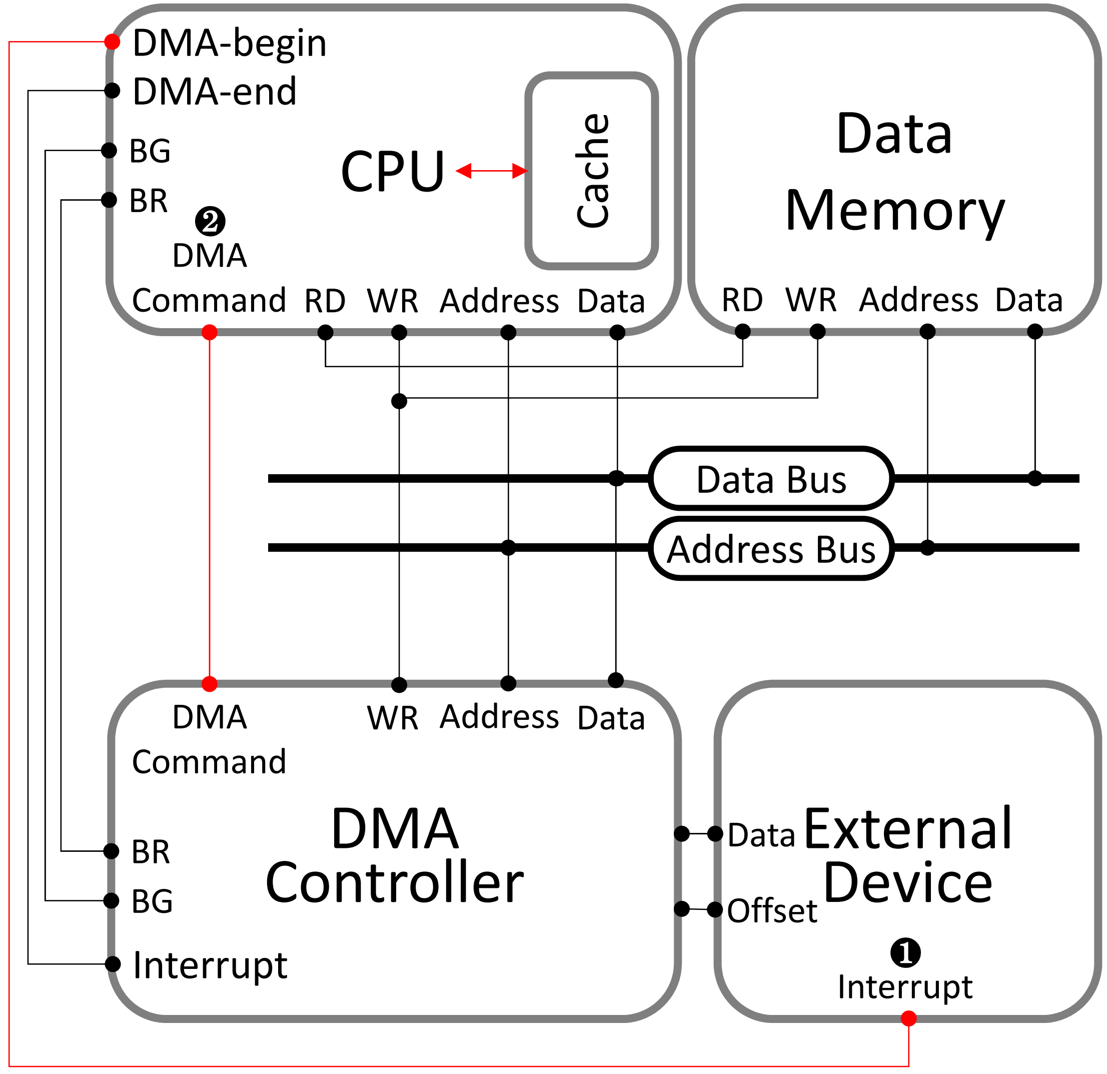
Simplified DMA

- We will use a simple DMA engine for educational purpose.
- Actual DMA is very complicated.
  - CPU generates commands for variable address and length
  - A device can read/write from/to the memory.
  - Multiple devices can be involved.
  - The device interrupts happen when the data transfer is done, and does not when the initiation.
  - The device writes the data and knows where to write.
  - …

Simulation Environment

- A virtual external device (external_device.v)
  - An external device has randomly-generated 12-words data.
  - When the device wants to write the data in the memory, it interrupts the CPU.
- Four-words bandwidth memory
  - Three memory transactions are required. (i.e., 4 word write x 3 times).
- The DMA controller arbitrates data transfer between memory and the device.

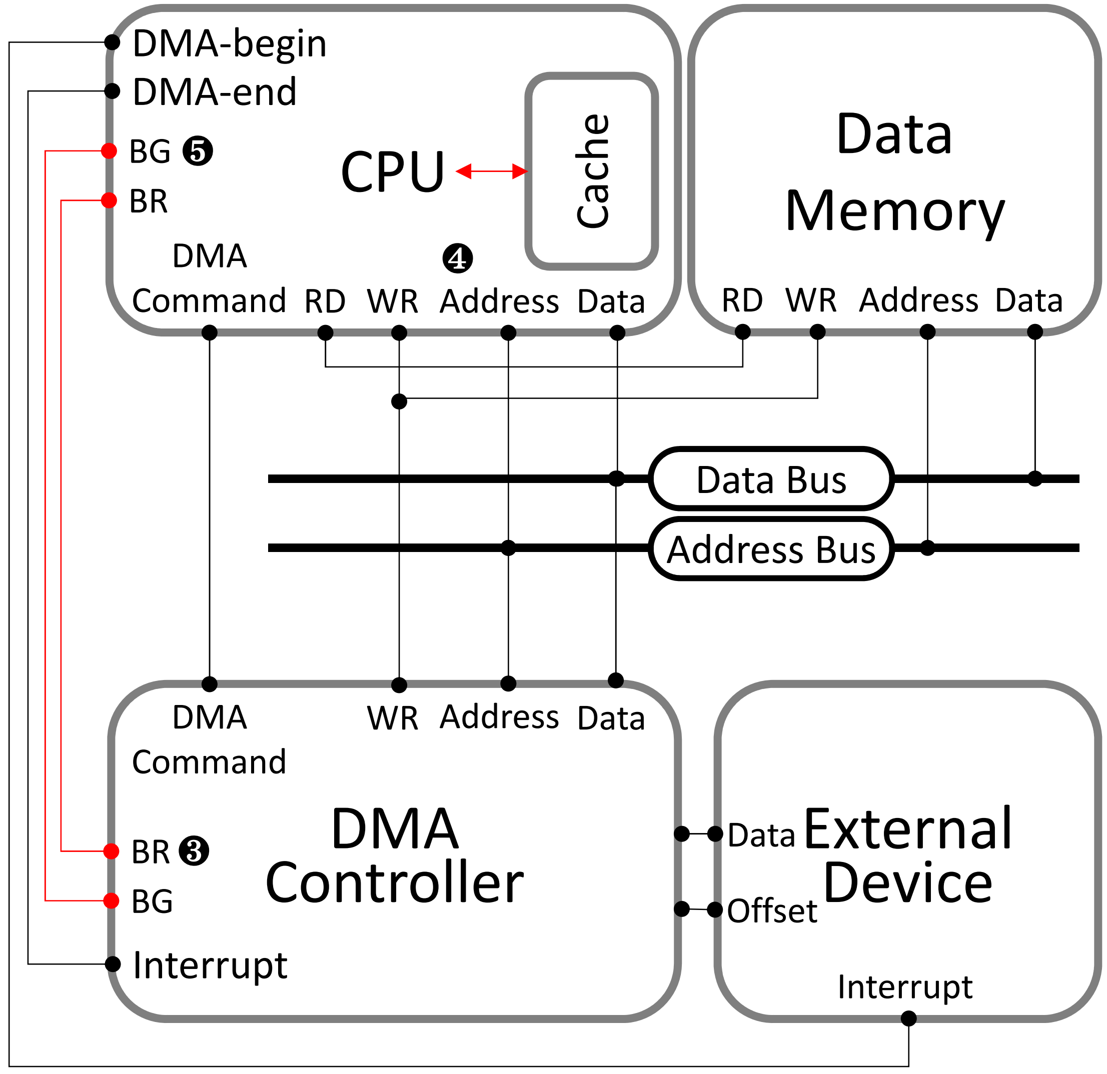
DMA Read/Write Transaction



1. The **external device** generates an interrupt. The **CPU** must be ready for handling DMA-begin and DMA-end interrupts.

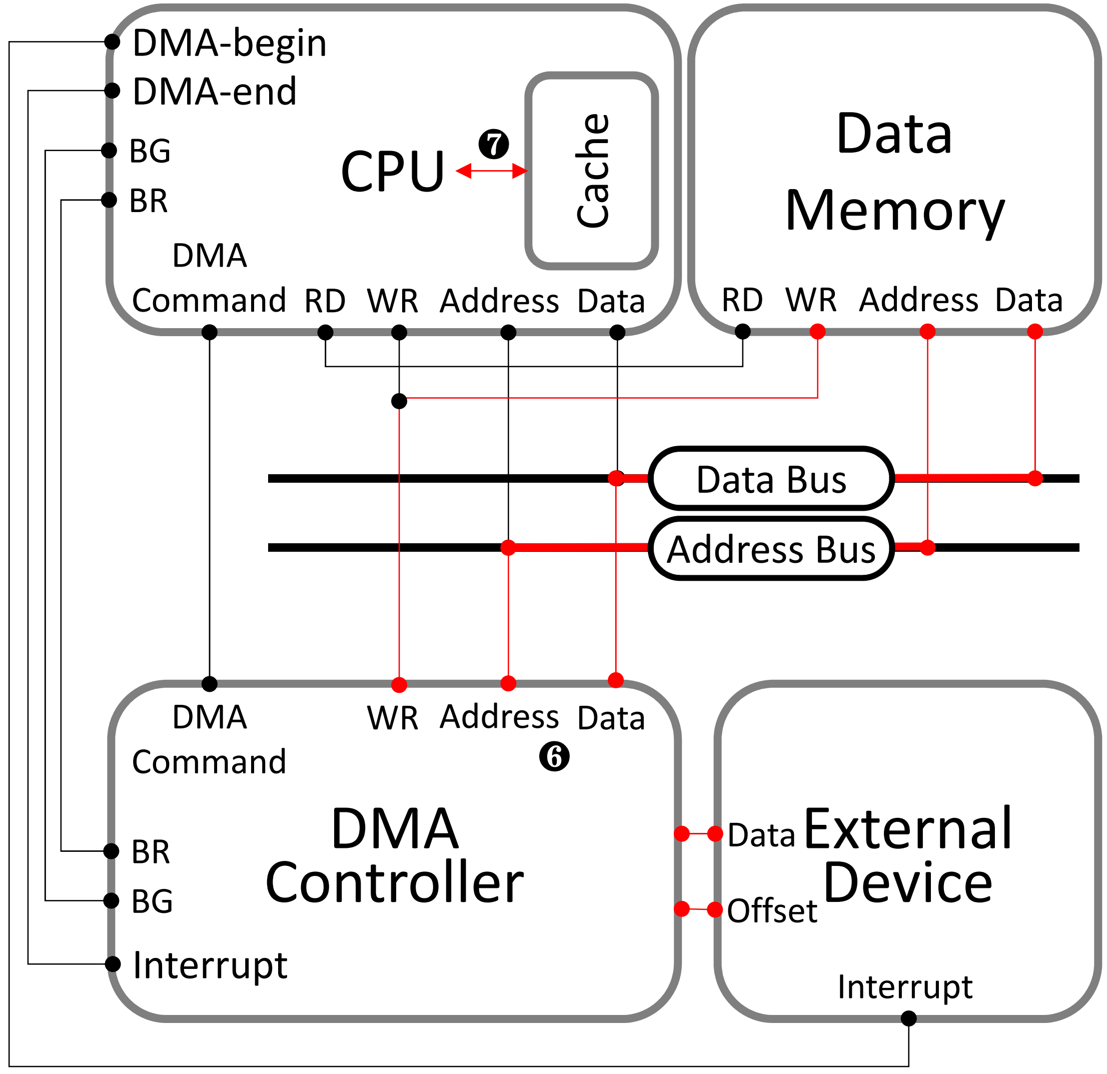
2. The **CPU** sends a command (address, length) to the **DMA controller**.
   - Address: the target memory address of DMA transactions, **0x1F4 (fixed address in this project)**
   - Length: the length of data, **12 words (fixed length in this project)**

DMA Read/Write Transaction



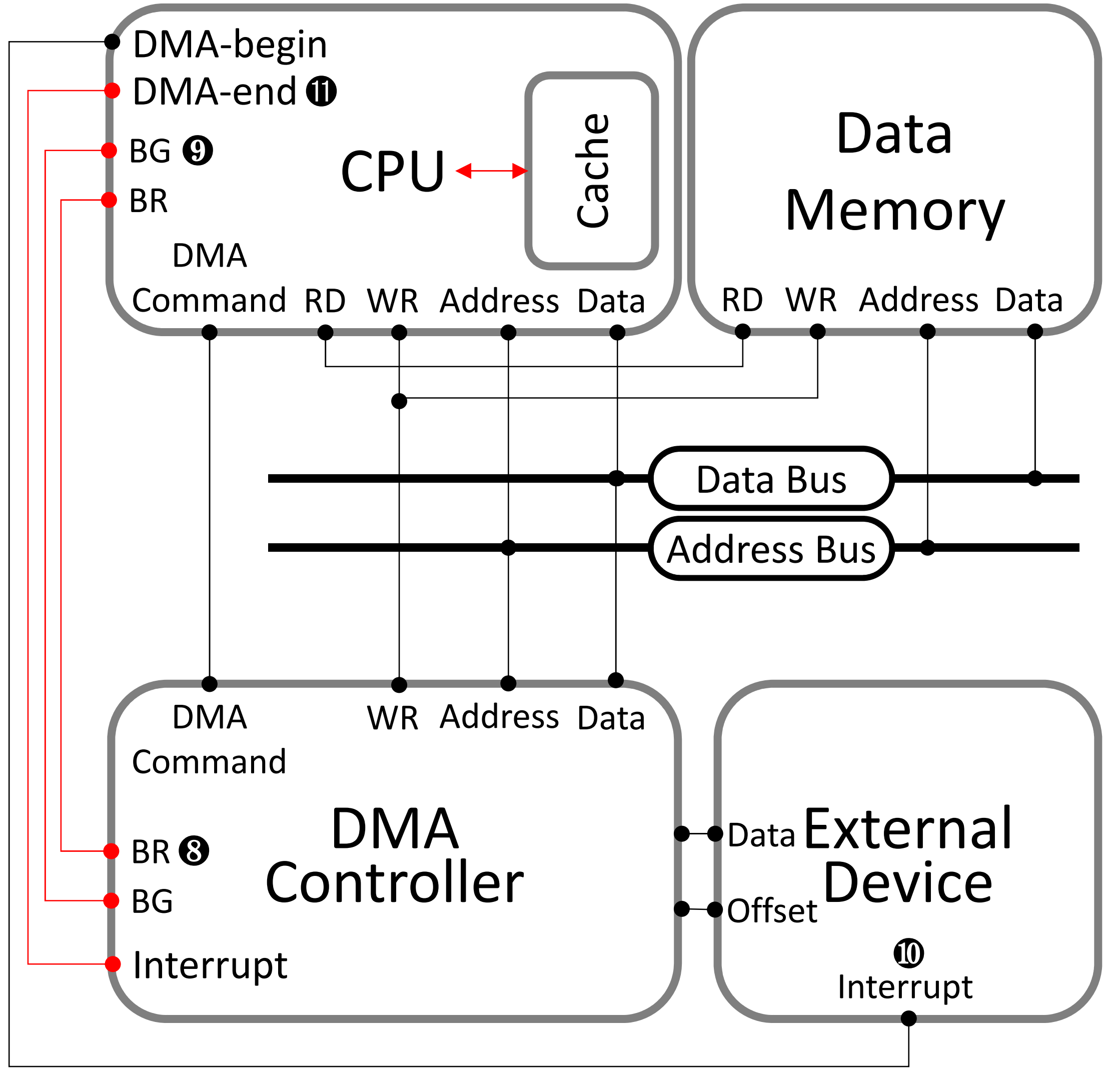
3. The **DMA controller** raises the Bus Request (BR) signal.
4. When it receives the BR signal, the **CPU** must stop using address/data buses and RD/WR signals of **data memory or ports (i.e., d_readM, d_writeM, d_address, d_data)**.
5. After that, the **CPU** raises the Bus Granted (BG) signal.

DMA Read/Write Transaction



6. After receiving the memory bus grant, the **DMA controller** performs DMA write transactions.
   - DMA write: the DMA controller read 12-word data from the external device and writes data to the target address.
7. At that time, **CPU** should run with only its cache. It should be blocked when a cache miss occurs.

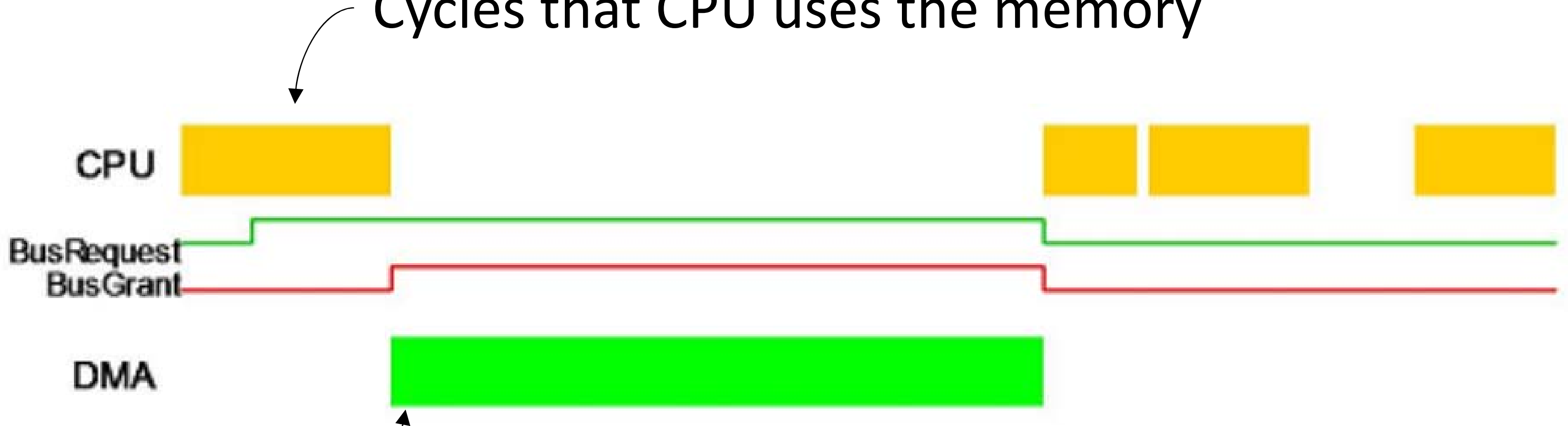
DMA Read/Write Transaction



8. The **DMA controller** clears the BR signals when it finishes data transfers between the device and memory.
9. The **CPU** clears the BG signal and enables using memory buses.
10. The **DMA controller** generates the DMA-end interrupt
11. The **CPU** handle the interrupt.

Example Simulation Results

Cycles that CPU uses the memory



Cycles that the DMA controller uses the memory

Required Implementation - CPU

- Interrupt handler
  - Two interrupt pins: DMA-begin, DMA-end
  - The CPU must stop and handle the interrupts immediately.
- DMA command generation
  - After receiving the DMA-begin interrupt, the CPU generates a DMA command and sends it to the DMA.
- Data memory bus arbitration
  - The DMA controller should be granted data memory buses when it raises the Bus Request signal.
  - The CPU cannot use data memory when the DMA controller is using the memory buses. However, the CPU can access both instruction and data caches.
  - After the DMA controller releases the memory buses, the CPU processes stalled memory operations immediately.
  - *Instruction memory or instruction ports are not involved in DMA in this project.*

Required Implementation - DMA

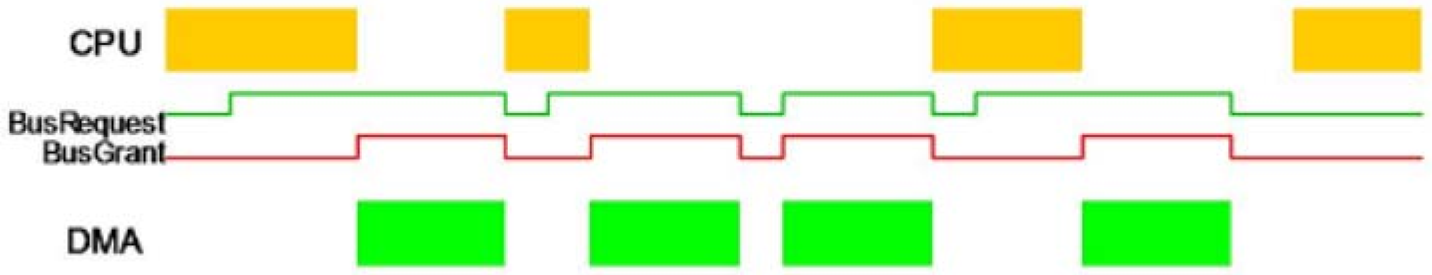
- DMA Controller
  - The DMA controller should receive and parse a DMA command from the CPU, and get the memory buses exclusively using Bus Request and Bus Granted signals.
  - After that, the DMA controller read the data from the external device and writes the data to the memory.
  - The DMA controller releases the data memory buses after all DMA transactions are done.
  - *DMA target address (0x1F4) and length (12 words) are fixed in this project.*

Grading

- Be still functional!
  - Your CPU should pass every test in the previous testbench.
  - Its functional correctness must not be affected by the DMA.
- Handle external signals and interrupts properly at the right time.
- Be working while the DMA is on-going unless the cache miss happens.
- After the DMA-end interrupt, the device's data should be read from the designated memory address.

Extra Credit: Cycle Stealing

- After send 4 words, DMA engine releases the bus.
- Then, the stalled-CPU can execute instruction.
- If the CPU does not use the memory bus, the DMA controller should retake the bus as soon as possible.
- Compare the performance to the case without cycle stealing.



Summary

- Understand what is DMA.
- Implement the simplified DMA logics in Verilog.
- Be careful about the required timing constraints.
- (Optional) Cycle stealing